Express

Infraestructura web rápida, minimalista y flexible para Node js.

Aplicaciones Web: Express es una infraestructura de aplicaciones Node js mínima y flexible que proporciona un conjunto sólido de características para las aplicaciones web.

API: Con miles de metodos de programa de utilidad HTTP y middleware a su disposición, la creación de una API sólida, rápida y sencilla.

Instalación

En la consola:

- npm install --save express

En el archivo .js.

```
var express = require('express');
var app = express();
```

Direccionamiento

El ***direccionamiento*** hace referencia a la determinación de cómo responde una aplicación a una solicitud de cliente en un determinado punto final, que es un URI (o una vía de acceso) y un método de solicitud HTTP específico (GET, POST, etc.).

Cada ruta puede tener una o varias funciones de manejador, que se excluyen cuando se correlaciona la ruta.

Métodos HTTP Soportados por Express

- GET (Obtiene un recurso)
- POST (Crea un nuevo recurso)
- PUT (Modifica un recurso)
- DELETE (Borra un recurso)
- HEAD
- OPTIONS
- TRACE
- COPY
- LOCK
- MKCOL
- MOVE
- PURGE
- PROPFIND
- UNLOCK
- REPORT

La definición de ruta tiene la siguiente estructura:

```
app.METHOD(PATH, HANDLER)
```

Donde:

- *app* es una instancia de *express.*
- *METHOD* es un método de solicitud *HTTP.*
- *PATH* es una vía de acceso a un servidor
- *HANDLER* es la función que se ejecuta cuando se correlaciona la ruta.

Ejemplo Ruta Básica

```
var express = require('express');
var app = express();

// GET method route
app.get('/', function(req, res) {
  res.send('hello world');
});
// POST method route
app.post('/', function (req, res) {
  res.send('POST request to the homepage');
});
```

Manejadores de Rutas

Los manejadores de rutas pueden tener la forma de una función, una matriz de funciones o combinaciones de ambas.

Una función de devolución de llamada individual puede manejar una ruta.

```
app.get('/example/a', function (req, res) {
  res.send('Hello from A!');
});
```

Más de una función de devolución de llamada puede manejar una ruta.

```
app.get('/example/b', function (req, res, next) {
  console.log('the response will be sent by the next function ...');
  next();
}, function (req, res) {
  res.send('Hello from B!');
});
```

Vías de Acceso de Ruta

Esta vía de acceso de ruta coincidirá con las solicitudes a la ruta raíz

```
app.get('/', function (req, res) {
  res.send('root');
});
```

Esta vía de acceso de ruta coincidirá con las solicitudes a /about.

```
app.get('/about', function (req, res) {
  res.send('about');
});
```

Esta vía de acceso de ruta coincidirá con las solicitudes a /random.text.

```
app.get('/random.text', function (req, res) {
  res.send('random.text');
});
```

Manejadores de Rutas

Existe un método de direccionamiento especial que no se deriva de ningún método HTTP.

En el siguiente ejemplo, el manejador se ejecutará para las solicitudes a “/secret”, tanto si utiliza GET, POST, PUT, DELETE, como cualquier otro método HTTP.

```
app.all('/secret', function (req, res, next) {
  console.log('Accessing the secret section ...');
  next(); // pass control to the next handler
});
```

Una matriz de funciones de devolución de llamada puede manejar una ruta.

```
var cb0 = function (req, res, next) {
  console.log('CB0');
  next();
}
var cb1 = function (req, res, next) {
  console.log('CB1');
  next();
}
var cb2 = function (req, res) {
  res.send('Hello from C!');
}

app.get('/example/c', [cb0, cb1, cb2]);
```

Métodos de Respuesta:

METODO	DESCRIPCIÓN
res.download()	Solicita un archivo para descargarlo.
res.end()	Finaliza el proceso de respuesta.
res.json()	Envía una respuesta JSON.
res.jsonp()	Envía una respuesta JSON con soporte JSONP.
res.redirect()	Redirecciona una solicitud.
res.render()	Representa una plantilla de vista.
res.send()	Envía una respuesta de varios tipos.
res.sendFile()	Envía un archivo como una secuencia de octetos.
res.sendStatus()	Establece el código de estado de la respuesta y envía su representación de serie como el cuerpo de respuesta.

Manejadores de Rutas Encadenables

Puede crear manejadores de rutas encadenables para una vía de acceso de ruta utilizando app.route(). Como la vía de acceso se especifica en una única ubicación, la creación de rutas modulares es muy útil, al igual que la reducción de redundancia y errores tipográficos.

```
app.route('/book')
  .get(function(req, res) {
    res.send('Get a random book');
  })
  .post(function(req, res) {
    res.send('Add a book');
  })
  .put(function(req, res) {
    res.send('Update the book');
  });
```

Recibir Parámetros GET

```
var express = require('express');
var app = express();

app.get('/user/:id', function (req, res) {
  res.send(req.params.id);
});

app.listen(3000);
```

Recibir Parámetros POST

```
var express = require('express');
var bodyParser = require(body-parser');
var app = express();

app.use(bodyParser.urlencoded({ extended: false}));
app.use(bodyParser.json());

app.post('/user', function (req, res) {
  res.send(req.body.id);
});
app.listen(3000);
```

Funciones Middleware

Las funciones middleware son funciones que tienen acceso a los objetos de solicitud y respuesta “req” y “res” y a la siguiente función en el ciclo solicitud/respuesta de la aplicación.

Las funciones pueden realizar las siguientes tareas:

- Ejecutar Cualquier código.
- Realizar cambios en la solicitud y los objetos de respuesta.
- Finalizar el ciclo de solicitud/respuesta.
- Invocar al siguiente middleware de la pila.

Si la función no finaliza el ciclo de solicitud/respuesta, se debe invocar next() para pasar el control a la siguiente función.

Desarrollo de Función Middleware

Para cargar la función de middleware, se llama a app.use(), especificando la función.

```
var express = require('express');
var app = express();


var myLogger = function (req, res, next) {
  console.log('LOGGED');
  next();
};


app.use(myLogger);

app.get('/', function (req, res) {
  res.send('Hello World!');
});

app.listen(3000);
```

El siguiente ejemplo añade una propiedad al objeto de solicitud.

```
var express = require('express');
var app = express();

var requestTime = function (req, res, next) {
  req.requestTime = Date.now();
  next();
};
app.use(requestTime);

app.get('/', function (req, res) {
  var responseText = 'Hello World!';
  responseText += 'Requested at: ' + req.requestTime + '';
  res.send(responseText);
});
app.listen(3000);
```

Tipos de Middleware

Una aplicación Express puede utilizar los siguientes tipos de middleware:

- Middleware de nivel de aplicación.
- Middleware de nivel de direccionador.
- Middleware de manejo de errores.
- Middleware incorporado.
- Middleware de terceros.

Middleware Incorporado

La única función de middleware incorporado en Express es express.static. Esta función es responsable del servicio de activos estáticos de una aplicación Express.

```
app.use('/miApp', express.static('./public'));
```

Middleware a Nivel de Direccionador

El middleware de nivel de direccionador funciona de la misma manera que el middleware de nivel de aplicación, excepto que está enlazado a una instancia de express.Router().

```
var express= express();
var router = express.Router();


// a middleware function with no mount path. This code is executed for every request to the router
router.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
});
```

```
// a middleware sub-stack shows request info for any type of HTTP request to the /user/:id path
router.use('/user/:id', function(req, res, next) {
  console.log('Request URL:', req.originalUrl);
  next();
}, function (req, res, next) {
  console.log('Request Type:', req.method);
  next();
});
```

```
// handler for the /user/:id path, which renders a special page
router.get('/user/:id', function (req, res, next) {
  console.log(req.params.id);
  res.render('special');
});

// mount the router on the app
app.use('/', router);
```